

# PHP-Schnittstelle zu MySQL

---

- **Queries mit Daten aus Parametern**

- Das folgende PHP-Script erzeugt einen Query-String, der von der Eingabe abhängt
  - Wurde kein Suchausdruck angegeben, werden alle Studenten angezeigt
  - Wurde ein Suchausdruck angegeben, werden alle Datensätze angezeigt, bei denen der Ausdruck im Namen vorkommt.

```
<?php
    $search = @$_GET['name'];
    // ...
    // Query stellen ...
    $query = "SELECT * FROM Student";
    if ($search)
        $query .= " WHERE Name LIKE '%$search%'";
    $res = mysqli_query($mysqli, $query);
```

- **Verständnisfrage:** ist Ihnen klar, wie diese Suche funktioniert?
  - Überlegen Sie ggf. wie der Query-String aussieht und wie LIKE funktioniert

# PHP-Schnittstelle zu MySQL

- Queries mit Daten aus Parametern

```
<?php $search = @$_GET['name']; ?>

<h1>Search for Student</h1>
<form action="" method=get>
    <input type=text      name=name      value="<?php print $search; ?>">
    <input type=submit    name=search    value="Search" >
</form>

<h1>Search Result
    <?php if ($search) print "for '$search'"; ?>
</h1>

<?php
    $mysqli = mysqli_connect(...);

    // Query stellen ...
    $query = "SELECT * FROM Student";
    if ($search)
        $query .= " WHERE Name LIKE '%$search%'";
    $res = mysqli_query($mysqli, $query);

    // Datensätze abrufen und verarbeiten
    print "<table>\n<tr><th>MatrNr <th>Name";
    while ($row = mysqli_fetch_assoc($res)) {
        print "\n<tr>";
        print "<td>" . $row['Name'];
```

# PHP-Schnittstelle zu MySQL

- **Queries mit Daten aus Parametern (Angriffe)**

- **Beispiel:** Suchausdruck „au“

- → `SELECT * FROM Student WHERE Name LIKE '%au%'`
  - Korrekt, liefert alle Studenten, deren Name „au“ enthält

- **Beispiel:** Suchausdruck „'“ (das Apostroph-Zeichen)

- → `SELECT * FROM Student WHERE Name LIKE '%''%'`
  - Der Query-String ist offensichtlich nicht syntaktisch korrekt
  - Es entsteht eine Fehlermeldung aus der Datenbank

- **Beispiel:** Suchausdruck „' AND MatrNr > 27000 ' AND '%' = '“

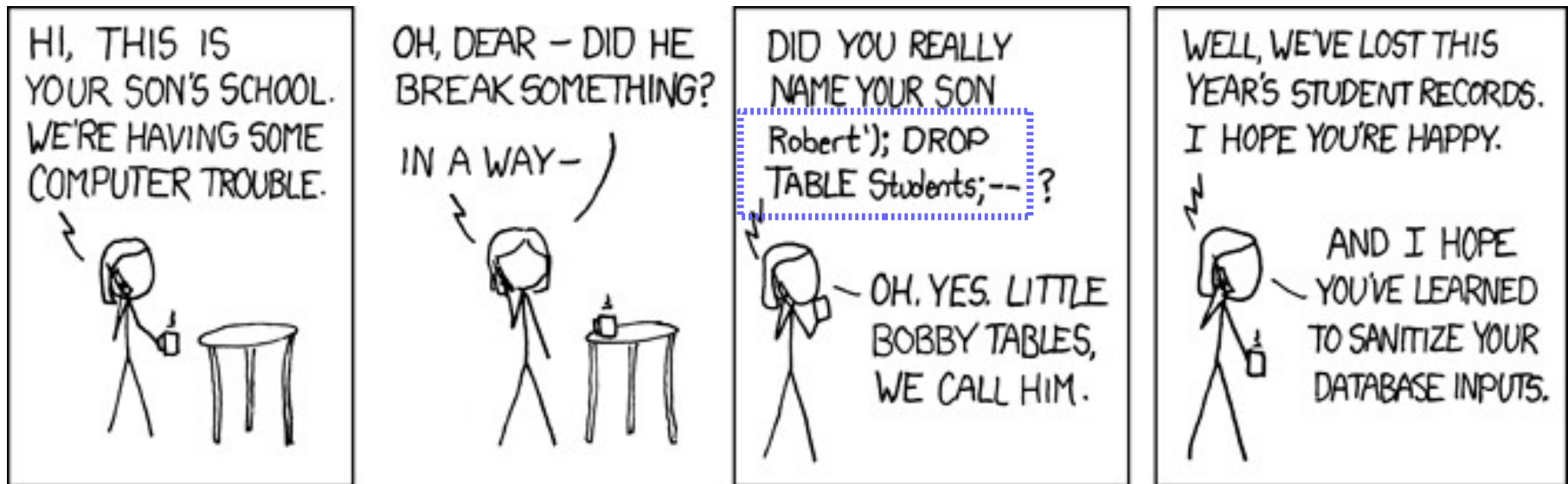
- → `SELECT * FROM Student WHERE  
Name LIKE '%' AND MatrNr > 27000 ' AND '%' = '%'`
  - Der Query-String ist syntaktisch korrekt, bedeutet aber etwas anderes
  - Der Ausdruck „'%' = '%‘“ liefert immer True und dient nur dazu, das nachfolgende „%'“ syntaktisch korrekt „unterzubringen“.

→ Ein Teil der **SQL-Instruktion** stammt **aus Nutzerdaten!**

- **Das nennt man SQL-Injection**

# Injects

- **XKCD-Comic 327** (alias „*Little Bobby Tables*“)



<https://xkcd.com/327/>

## – Übung:

- Wie sah das gesamte SQL-Statement hier also (vermutlich) aus?
- Was soll es bewirken?
- Lösung: [https://www.explainxkcd.com/wiki/index.php/327:\\_Exploits\\_of\\_a\\_Mom](https://www.explainxkcd.com/wiki/index.php/327:_Exploits_of_a_Mom)

# Injections

---

- **SQL-Injection**

- Bei einer SQL-Injection stammen nicht nur (wie beabsichtigt) Daten im Query-String, sondern auch **SQL-Statements, Bedingungen, etc. aus unzuverlässigen Quellen** (z.B. vom Endanwender)
- Ursache: Bei der Bildung des Query-Strings werden SQL-Statements und Benutzer-Daten zu einem Query-String kombiniert.
  - Wenn bei der späteren Zerlegung und Ausführung des entstandenen Strings eine **andere Interpretation** entsteht als ursprünglich beabsichtigt, können Benutzerdaten fälschlich als SQL-Statements interpretiert werden
  - Siehe auch <http://de.wikipedia.org/wiki/SQL-Injection>

- **Mögliche Folgen**

- Syntaktische Fehler beim Datenbankzugriff
  - Dadurch kann man evtl. Abläufe stören (z.B. Logging von Ereignissen)
- Manipulation von Auswahl-Bedingungen (WHERE-Bedingungen)
  - Dadurch kann man z.B. fremde Datensätze sehen
- Einschleusung kompletter Datenbank-Anweisungen
  - Dadurch kann man evtl. beliebige Datenbank-Operationen vornehmen (z.B. Passwörter ändern)

# Injectons

- **Mögliche Folge: Ausweitung von Ergebnis-Menge**

- Dadurch kann man z.B. fremde private Datensätze sehen

- **Beispiel:**

- Ein eingeloggter Student soll die von ihm selbst (MatrNr aus Session) belegten Vorlesungen nach Titeln durchsuchen können.

```
<?php
$search = @$_GET['titel'];
$matnr = @$_SESSION['MatrNr']; // gesicherte MatrNr des eingeloggten Stud.
// ...
$query = "SELECT * FROM hört JOIN Vorlesung USING (VorlNr) WHERE";
if ($search)
    $query .= " Titel LIKE '%$search%' AND";
$query .= " MatrNr=$matnr";
```

- Beabsichtigte Query (Suchausdruck „ET“):
      - SELECT \* FROM hört JOIN Vorlesung USING (VorlNr)  
WHERE Titel LIKE '%ET%' AND MatrNr=25403

Liefert nur  
eigene Vorlesungen

- Query durch Injection-Suchausdruck „' -- '“
      - SELECT \* FROM hört JOIN Vorlesung USING (VorlNr)  
WHERE Titel LIKE '% ' -- %' AND MatrNr=25403

Liefert Vorlesungen  
aller Studenten

“ -- “  
(mit nachfolgendem  
Leerzeichen!)  
macht in SQL den  
Rest der Zeile zum  
Kommentar.

# Injects

- **Mögliche Folge: Einschleusen neuer SQL-Anweisungen**

- Dadurch kann man beliebige Manipulationen in der Datenbank vornehmen (im Rahmen der Rechte des Benutzers)
- **Beispiel** (zu Studenten-Suchfunktion von oben):

- **Injection-Suchausdruck**

`' ; DELETE FROM Professor -- '“`

- **Resultierende Anfrage:**

`SELECT * FROM Student WHERE Name LIKE '%';  
DELETE FROM Professor -- '%'`

- Löscht alle Professor-Datensätze

- **Analog:** ändern von Passwörtern, exmatrikulieren von Studierenden, ...

- Dieses Beispiel funktioniert bei `mysqli_query` nicht, da hier keine **mehrteiligen SQL-Queries** erlaubt sind (→ dann Syntax-Fehler)

# Injections

- **Gegenmaßnahmen zu SQL-Injections (1)**

- Daten im Query-String vor Einbettung geeignet „**escapen**“
  - Vermeidet die Unterbrechung des umgebenden ' ... ' -Strings
  - Funktion: **mysqli\_real\_escape\_string** ( mysqli \$link , string \$escapestr )
    - Behandelt **NUL** (ASCII 0), **\n**, **\r**, **\\**, **'**, **"**, **Control-Z**.
    - Wird auf jeden Parameter einzeln angewandt
  - Beispiel:

```
<?php
$search = @$_GET['name'];
// ...
// Query stellen ...
$query = "SELECT * FROM Student";
if ($search) {
    $xsearch = mysqli_real_escape_string($mysqli, $search)
    $query .= " WHERE Name LIKE '%$xsearch%'";
}
$res = mysqli_query($mysqli, $query);
```

- **Verständnisfrage**: Kann ich den ganzen Query-String damit behandeln?

# Injectons

---

- **Gegenmaßnahmen zu SQL-Injections (2a)**

- Idee: Daten nicht in Query-String einbetten, sondern **separat** an das DBMS übergeben
- **Technik: Prepared Statements**
  - Ich formuliere zunächst den Query-String ohne Daten.
    - Er enthält überall ein „?“, wo Daten eingefügt werden sollen.
    - Beispiel: „**SELECT \* FROM Student WHERE a=? AND b=?**“
    - Ziel: Ein „prepared Statement“
  - Erst in einem späteren Aufruf „binde“ ich Parameter an das prepared Statement
    - Beides wird getrennt an das DBMS übergeben und verarbeitet.
    - Es es gibt also keine Vermischung von Code (SQL-Statements) und Daten

# Injections

- **Gegenmaßnahmen zu SQL-Injections (2b)**

- Daten nicht in Query-String einbetten, sondern **explizit** übergeben

- **Technik: Prepared Statements**

- Query-String enthält überall ein „?“, wo Daten eingefügt werden sollen.
- mit `mysqli_prepare()` wird ein Prepared Statement erzeugt
- mit `mysqli_stmt_bind_param()` werden die Parameterwerte gebunden
- mit `mysqli_stmt_execute()` wird der Query ausgeführt
- mit `mysqli_stmt_get_result()` wird das Ergebnis geholt

- **Beispiel:**

```
<?php
$a = @$_GET['a'];      $b = @$_GET['b'];
$stmt = mysqli_prepare($mysqli,
    "SELECT * FROM Student WHERE a=? AND b=?");
mysqli_stmt_bind_param($stmt, 'ss', $a, $b);
mysqli_stmt_execute($stmt);
$res = mysqli_stmt_get_result($stmt);

while ($row = mysqli_fetch_assoc($res)) {
    // ...
}
```

- Der 2. Parameter in `mysqli_stmt_bind_param()` gibt die Typen an
  - 's'=String, 'i'=integer, 'd'=Fließkommazahl, 'b'=BLOB

# Injections

---

- **Gegenmaßnahmen zu SQL-Injections (3)**
  - Keine mehrteiligen SQL-Statements in Query-Strings zulassen
    - Dies ist bei `mysqli_query` ja gesichert (s.o.)
    - Es bietet aber keine vollständige Sicherheit
  - Filterung von Eingaben
    - z.B. bekannte Injection-Muster erkennen und ablehnen oder bereinigen
    - Problem: **False Positives**
      - Zulässige Eingaben werden abgelehnt oder verändert
    - Problem: **False Negatives**
      - Unbekannte (neue / modifizierte) Angriffe werden nicht sicher erkannt
- **SQL-Injections können sehr vielseitig ablaufen**
  - Siehe auch <http://de.wikipedia.org/wiki/SQL-Injection>
    - Überschreiben von Server-Dateien mit INTO OUTFILE
    - Überlasten des Servers, löschen von Tabellen, Ändern von Rechten, ...

# Injections

---

- **Injections**

- Allgemein bezeichnet man mit „**Injection**“ jede Form von **fremdbestimmten Daten**, die als **Code** interpretiert werden
- „**Code**“ in diesem Sinne sind **alle Kontrollstrukturen**
  - also nicht nur klassische Programmiersprachen
  - **Beispiele:**
    - SQL-Statements,
    - Javascript-Code,
    - HTML-Tags und HTML-Tag-Parameter
- „**fremdbestimmte Daten**“ sind alle Daten, die von nicht vertrauenswürdiger Seite beeinflussbar sind
  - **Beispiele:**
    - Benutzereingaben (GET- oder POST-Parameter)
    - Cookies
    - HTML-Header
    - hochgeladene Dateien

# Injections

---

- **Injections (Beispiele)**

- Über einen POST-Parameter werden **SQL-Fragmente** übergeben, die eine Verfälschung der in der Folge erzeugten **SQL-Queries** verursachen
  - **SQL-Injection**
  - Gefahr: Fehler, Manipulation der Datenbank, Geheimnis-Enthüllung
  - Gegenmaßnahmen (s.o.):
    - Parameter-Escaping
    - sichere Query-Erzeugung
    - Daten-Filterung (evtl. unsicher)
- Das obige Script enthält eine solche Verwundbarkeit

# Injections

- Queries mit Daten aus Parametern

```
<?php $search = @$_GET['name']; ?>

<h1>Search for Student</h1>
<form action="" method=get>
    <input type=text      name=name      value="<?php print $search; ?>">
    <input type=submit    name=search    value="Search" >
</form>
<h1>Search Result
    <?php if ($search) print "for '$search'"; ?>
</h1>
<?php
    $mysqli = mysqli_connect(...);

    // Query stellen ...
    $query = "SELECT * FROM Student";
    if ($search)
        $query .= " WHERE Name LIKE '$search'";
    $res = mysqli_query($mysqli, $query);

    // Datensätze abrufen und verarbeiten
    print "<table>\n<tr><th>MatrNr <th>Name";
    while ($row = mysqli_fetch_assoc($res)) {
        print "\n<tr>";
        print "<td>" . $row['Name'];
```

# Injectons

---

- **Injectons (Beispiele)**

- Über einen GET-Parameter werden **HTML-Fragmente** übergeben, die eine Verfälschung der in der Folge erzeugten **HTML-Seite** verursachen.
- Diese HTML-Fragmente können auch **Javascript** enthalten, die im Browser eines Nutzers ausgeführt werden
  - **HTML-Injection**
  - **Javascript-Injection („XSS“ = Cross-Site-Scripting)**
  - Gefahr: Fehlerhafte Darstellung, Manipulation der Benutzer-Webseite, Geheimnis-Enthüllung durch Javascript (z.B. gefälschte Passwort-Dialoge)
  - Gegenmaßnahmen:
    - Parameter-Escaping (PHP-Funktion **htmlspecialchars**)
    - Daten-Filterung (z.B. PHP-Funktion **strip\_tags**) (allgemein evtl. unsicher)
- Das obige Script enthält zwei solcher Verwundbarkeiten

# Injects

- HTML-Injection, XSS

```
<?php $search = @$_GET['name']; ?>

<h1>Search for Student</h1>
<form action="" method=get>
    <input type=text      name=name      value="<?php print $search; ?>">
    <input type=submit    name=search    value="Search" >
</form>
<h1>Search Result
    <?php if ($search) print "for '$search'"; ?>
</h1>
<?php
    $mysqli = mysqli_connect(...);

    // Query stellen ...
    $query = "SELECT * FROM Student";
    if ($search)
        $query .= " WHERE Name LIKE '$search'";
    $res = mysqli_query($mysqli, $query);

    // Datensätze abrufen und verarbeiten
    print "<table>\n<tr><th>MatrNr <th>Name";
    while ($row = mysqli_fetch_assoc($res)) {
        print "\n<tr>";
        print "<td>" . $row['Name'];
```

# Injections

---

- **HTML-Injection, XSS (Demo)**

- **Beispiel:** Demo-XSS-Injection String für Such-Feld:

`" onclick="alert('Hallo')"`

Liefert auf dem verwundbaren Input-Feld:

`<input type="text" name="name" value="" onclick="alert('Hallo')">`

Öffnet also beim Klick auf das Text-Eingabefeld einen Alert.

- **Übung**
  - Überlegen Sie sich einen Injection-String, durch den Sie das Action-Feld des Formulars auf einen beliebige URL setzen können.  
(Zwei JS-Funktionsaufrufe genügen.)
- **Angriffsszenario** (bei nicht geschützten Servern)
  - Sie ergänzen die Login-URL einer Bank mit einem GET-Parameter, durch den die Login-Daten an einen fremden Server geschickt werden. Dann schicken Sie die URL per gefälschter Email an einen Kunden mit der Aufforderung, sich „*ganz dringend*“ darüber einzuloggen.

# Injectons

---

- **Persistente / Reflektierte Injectons**
  - Injectons müssen nicht immer unmittelbar durch einen Query erfolgen
  - Sie können auch über Daten erfolgen, die zunächst auf dem Server dauerhaft („**persistent**“) gespeichert werden und erst bei einer späteren Abfrage einen Effekt haben
    - Sie werden sozusagen vom Server „**reflektiert**“
  - Beispiel:
    - In der Datenbank liegt ein Fragment für eine Javascript-Injection (**XSS**)
    - In der Datenbank liegt ein Fragment für eine **SQL-Injection**
      - **Verständnisfrage**: Wie kann das sein? Sie wurde doch escaped beim Einfügen?
    - Über eine Upload-Funktion wird ein PHP-Script hochgeladen, dass vom Server beim späteren Zugriff ausgeführt wird (**Script-Injection**)
      - Gegenmaßnahme: Im Upload-Bereich immer aktives Scripting deaktivieren!
  - Das obige Script enthält eine solche Verwundbarkeit

# Injectons

- Persistente HTML-Injection, XSS

```
<?php $search = @$_GET['name']; ?>

<h1>Search for Student</h1>
<form action="" method=get>
    <input type=text      name=name      value="<?php print $search; ?>">
    <input type=submit    name=search    value="Search" >
</form>
<h1>Search Result
    <?php if ($search) print "for '$search'"; ?>
</h1>
<?php
    $mysqli = mysqli_connect(...);

    // Query stellen ...
    $query = "SELECT * FROM Student";
    if ($search)
        $query .= " WHERE Name LIKE '$search'";
    $res = mysqli_query($mysqli, $query);

    // Datensätze abrufen und verarbeiten
    print "<table>\n<tr><th>MatrNr <th>Name";
    while ($row = mysqli_fetch_assoc($res)) {
        print "\n<tr>";
        print "<td>" . $row['Name'];
```

# Injecti0ns

---

- **Persistente / Reflektierte Injecti0ns (Demo)**

- **Beispiel:** Demo-DB-XSS-Injection String für Studenten-Name:

```
INSERT INTO Student (Name) VALUES  
(' <script>alert("DB")</script>');
```

Öffnet beim Auflisten des Studenten in jeder Such-Ausgabe einen Alert.

- **Übung**

- Die DB-Textfelder sind evtl. sehr kurz für komplexeren JS-Code (Student.Name hier 64 Zeichen). Wie könnte man das umgehen?
- Können Sie sich ein Szenario vorstellen, in dem man ausdrücklich Daten aus der Datenbank ungeschützt ausgeben will?

- **Angriffsszenario** (bei nicht geschützten Servern)

- Sie schreiben in ein Forum einen Kommentar. Auf der Einstiegsseite des Forums werden die neuesten Kommentare (ungeschützt) angezeigt. Der JS-Code im Kommentar sendet jeweils das Session-Cookie als GET-Parameter an einen fremden Server.

# Injections

- **Abgesichertes Beispiel** → mysqli\_04\_search\_safe [Quelle](#) + [Ausführbar](#)

```
<!DOCTYPE html>
<html>
<head></head>
<body>
<?php $search = @$_GET['name']; ?>

<h1>Search for Student</h1>
<form action="" method=get>
    <input type=text      name=name      value="<?php print htmlspecialchars($search); ?>">
    <input type=submit    name=search    value="Search" >
</form>

<h1>Search Result
<?php if ($search) print "for '".htmlspecialchars($search)."'"; ?>
</h1>
<?php
    require('/home/lamp/.mysql_credentials');
    $mysqli = mysqli_connect($mysql_server, $mysql_user, $mysql_password, $mysql_database);
    if (!$mysqli) {
        echo "Failed to connect to MySQL: " . mysqli_connect_error();
        exit(1);
    }
    $query = "SELECT * FROM Student";
    if ($search) {
        $xsearch = mysqli_real_escape_string($mysqli, $search);
        $query .= " WHERE Name LIKE '%$xsearch%'";
        // $query .= " OR MatrNr LIKE '%$xsearch%'";
    }
    print '<div style="margin: 1em 0; padding: 0 1em; border: thin solid grey;">';
    print "<pre>" . htmlspecialchars($query) . "</pre>";
    $res = mysqli_query($mysqli, $query);
    if (!$res) {
        echo "Failed to Query MySQL: " . mysqli_error($mysqli);
        exit(1);
    }
    print "</div>";
    print "<table border=1>\n<tr><th>MatrNr <th>Name";
    while ($row = mysqli_fetch_assoc($res)) {
        print "\n<tr>";
        print "<td>" . htmlspecialchars($row['MatrNr']);
        print "<td>" . htmlspecialchars($row['Name']);
    }
    print "\n</table>";
    mysqli_close($mysqli);
?>
</body>
</html>
```